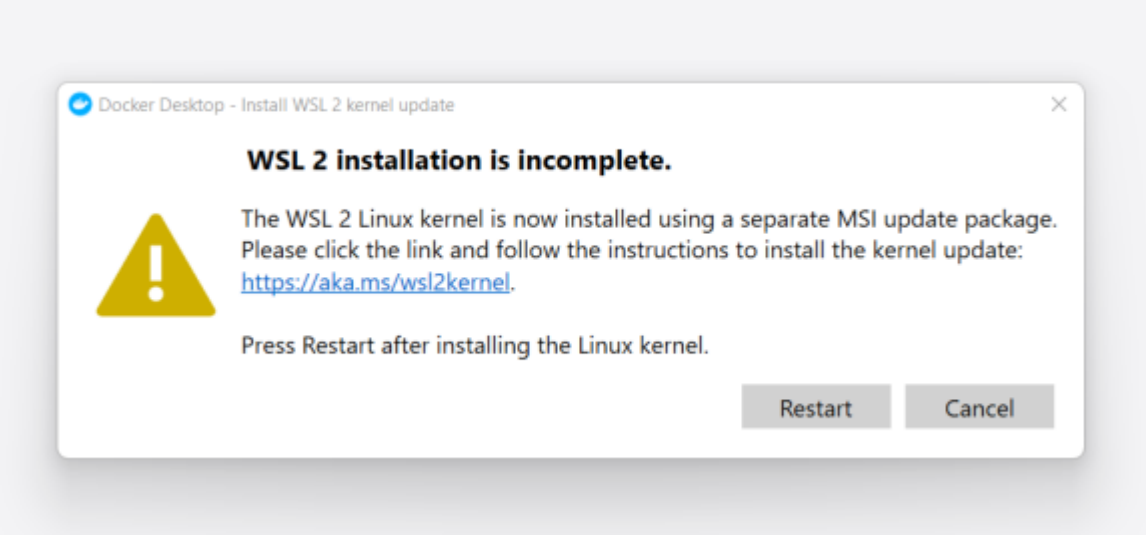
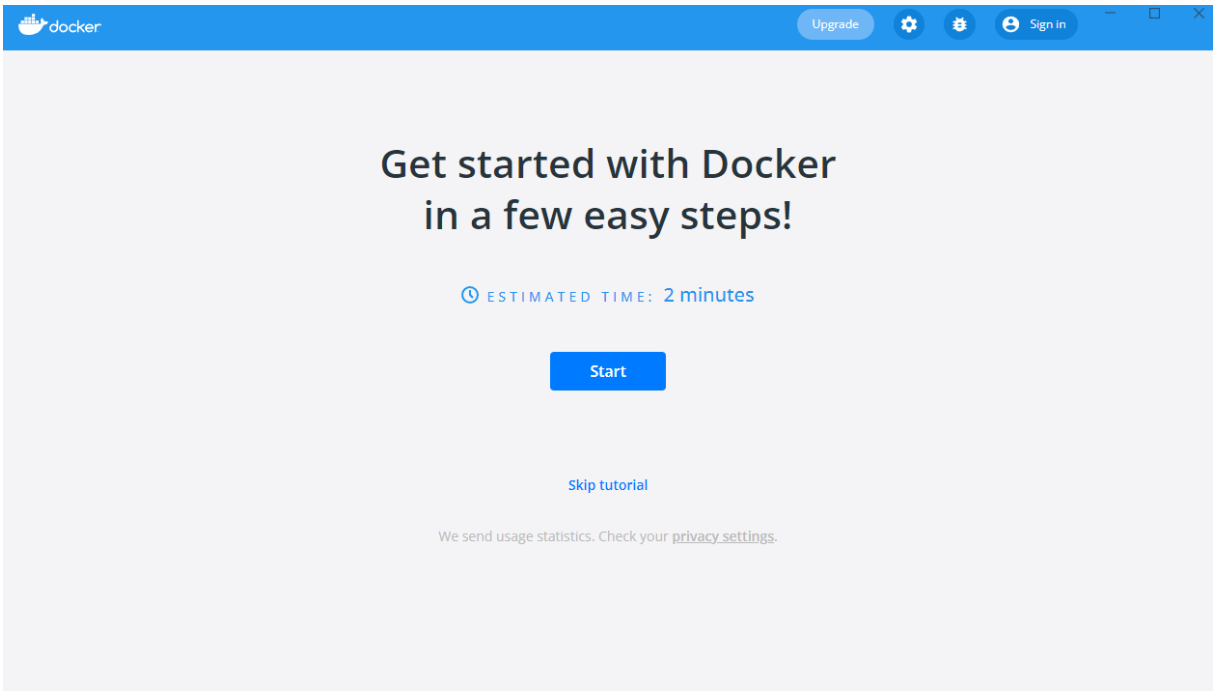
Sistemska administracija – 1. projektna naloga

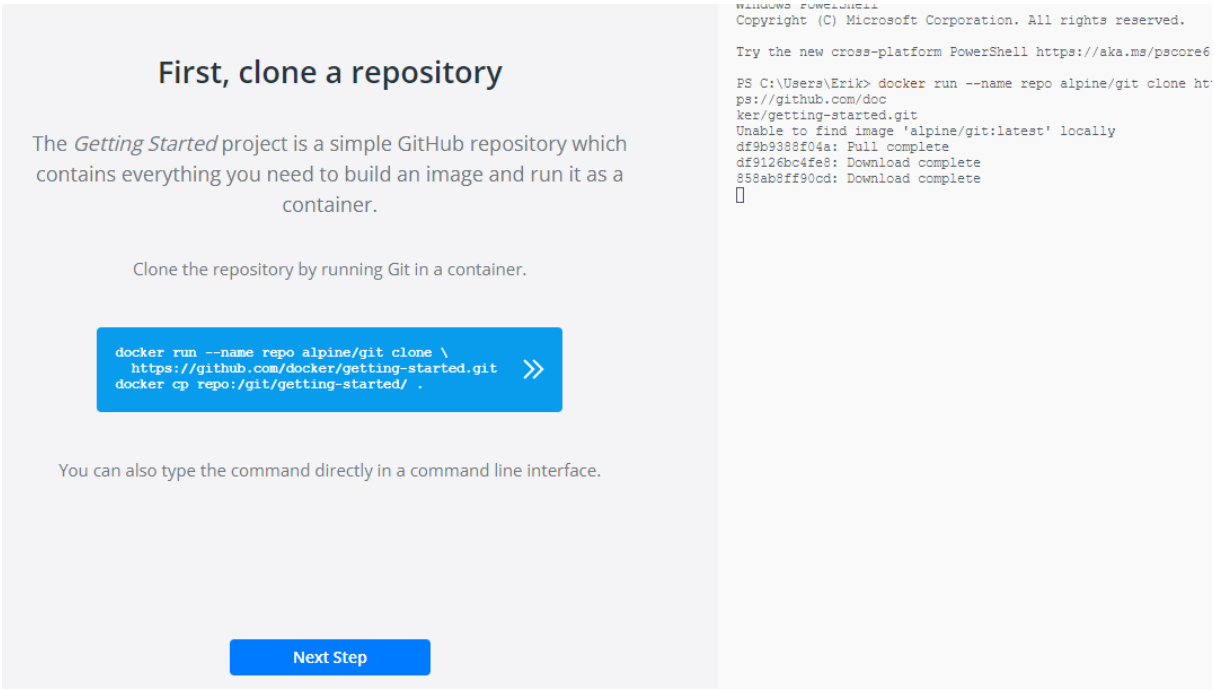
1. Nameščanje Docker-ja in vodič



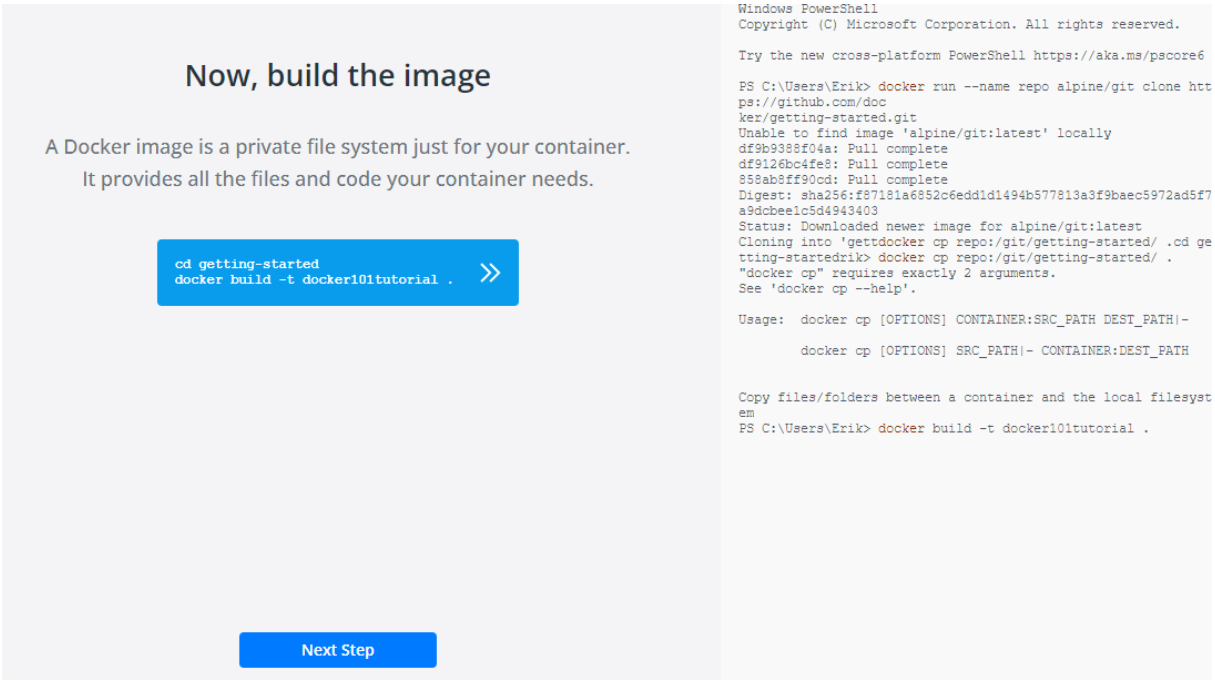
Takoj po prenosu Docker-ja smo naleteli na prvo težavo. Bilo je potrebno posodobiti verzijo WSL-ja. Na tem naslovu je dokaj enostaven vodič, kako ta problem rešiti. URL naslov za posodobitev WSL2 Po namestitvi WSL2 Linux kernel update paketa je bilo še potrebno v Windows PowerShell zapisati ukaz wsl –set-default-version 2. Za tem je Docker končno deloval.



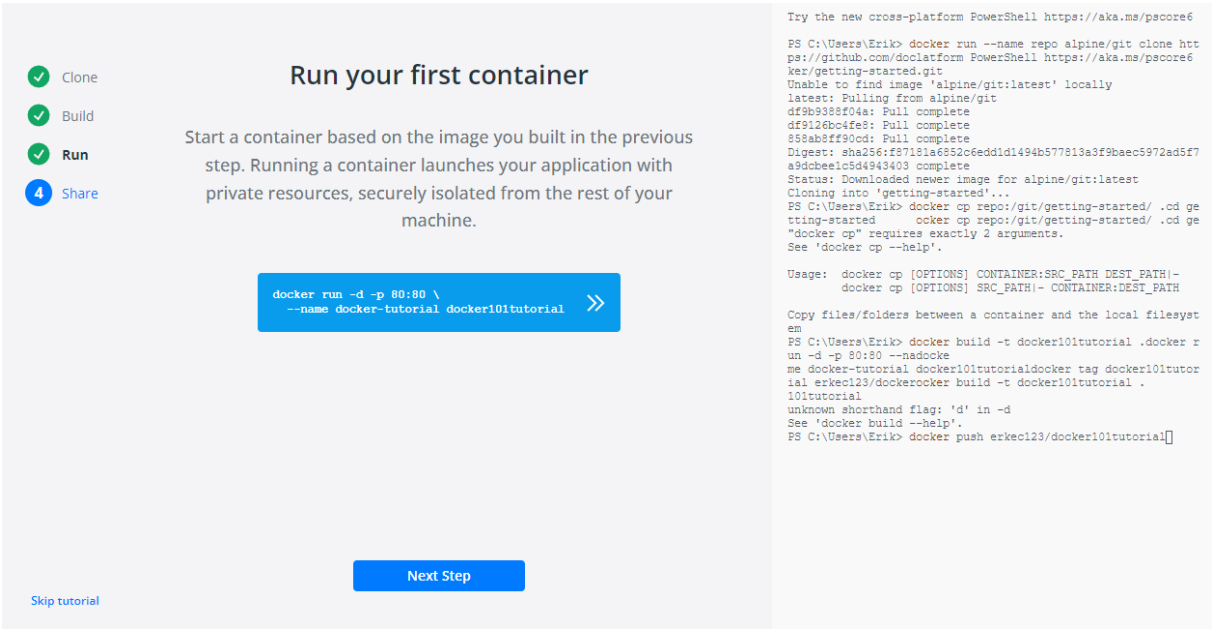
Prvo je potrebno klonirati repozitorij.



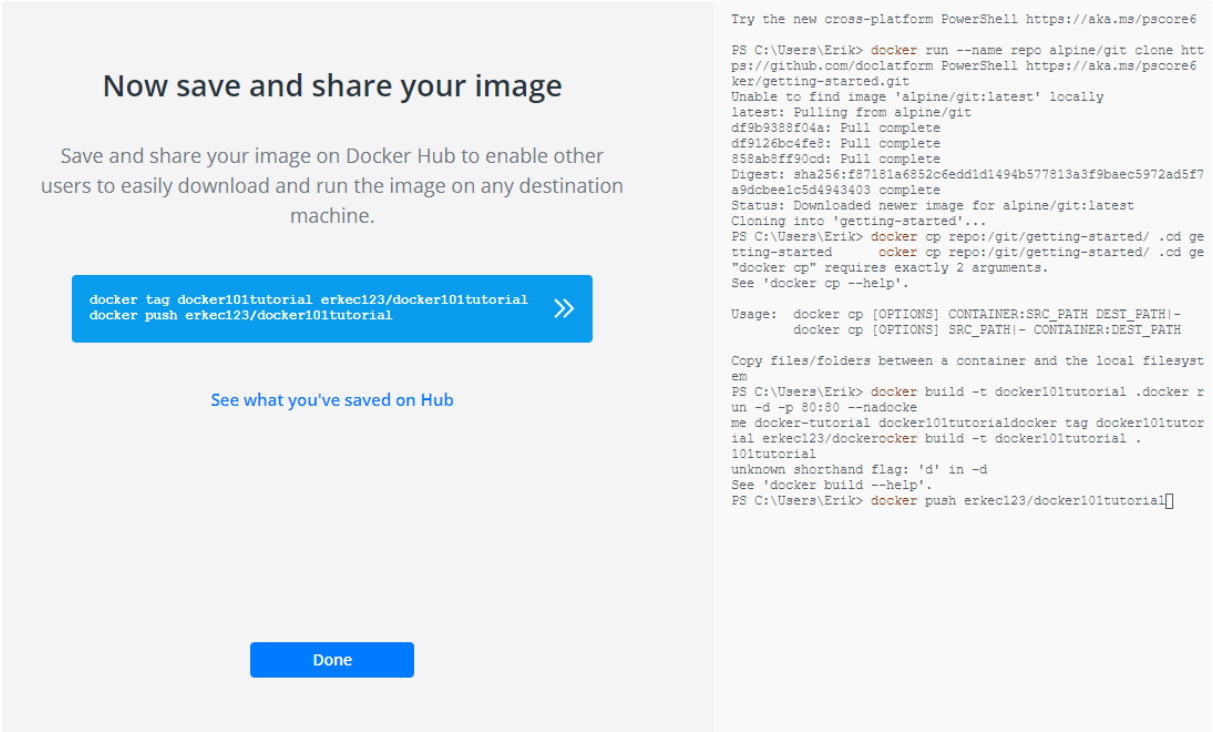
Nato je potrebno zgenerirati sliko. Ko zaženemo zaboj, uporabljamo izoliran datotečni sistem. Ta datotečni sistem po meri zagotavlja slika zaboja. Ker slika vsebuje datotečni sistem zaboja, mora vsebovati vse, kar je potrebno za zagon aplikacije – vse odvisnosti, konfiguracijo, skripte, binarne datoteke itd. Slika vsebuje tudi druge konfiguracije za zaboj, kot so spremenljivke okolja, privzeti ukaz za zagon, in drugi metapodatki.



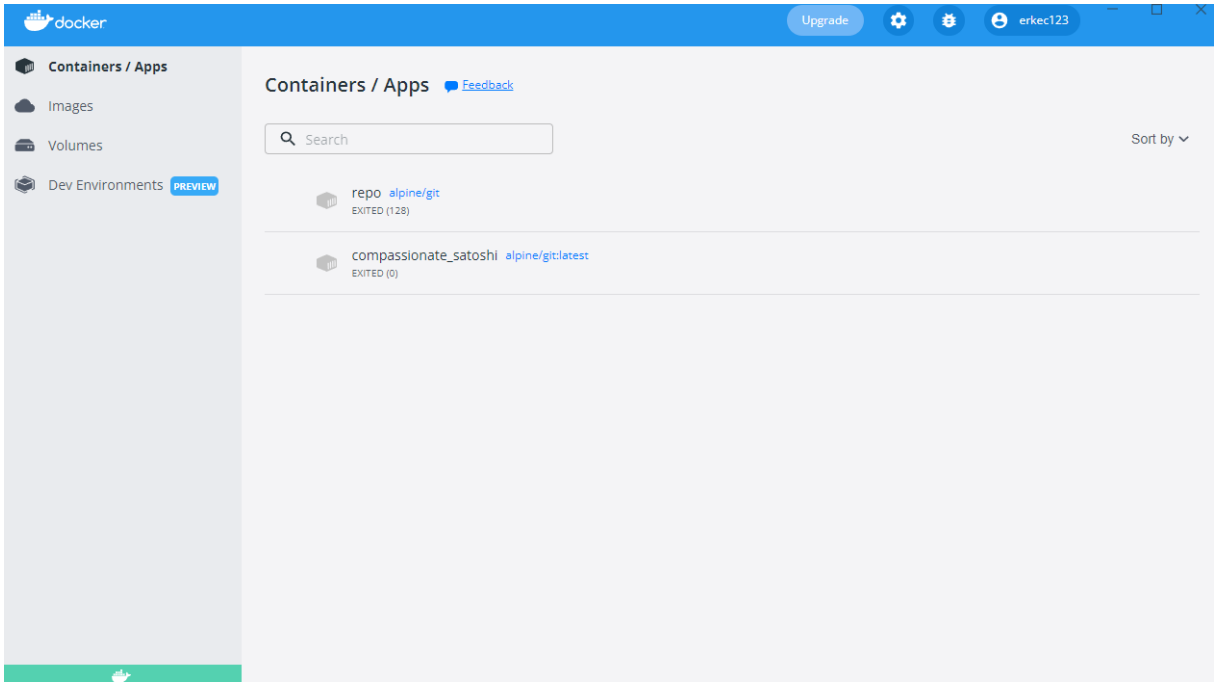
Končno lahko ustvarimo naš »container« oziroma zaboj, to je proces v peskovniku na vašem računalniku, ki je izoliran od vseh drugih procesov na gostiteljskem računalniku.



Ko smo končali s temi koraki, se je potrebno prijaviti in nato lahko shranimo in delimo našo sliko.

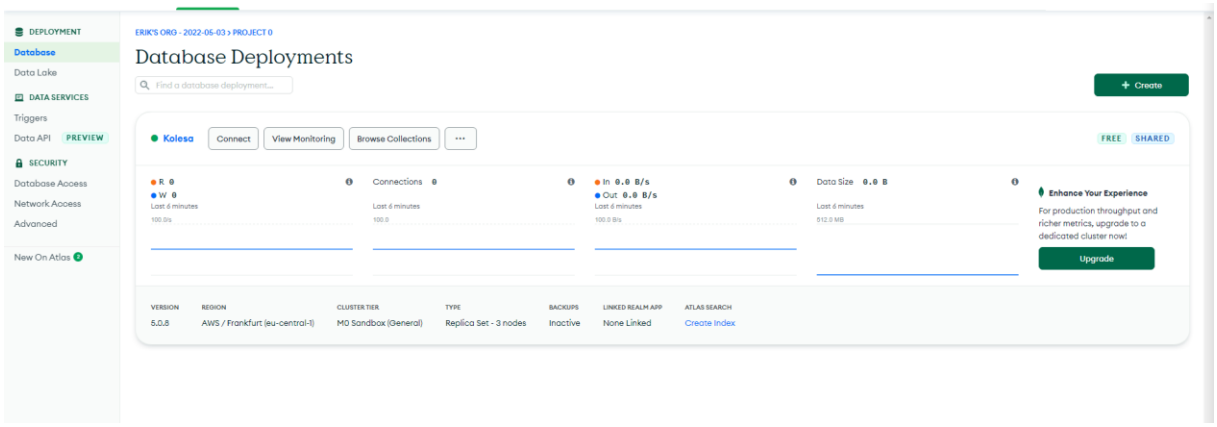


Tako zgleda naš Docker.

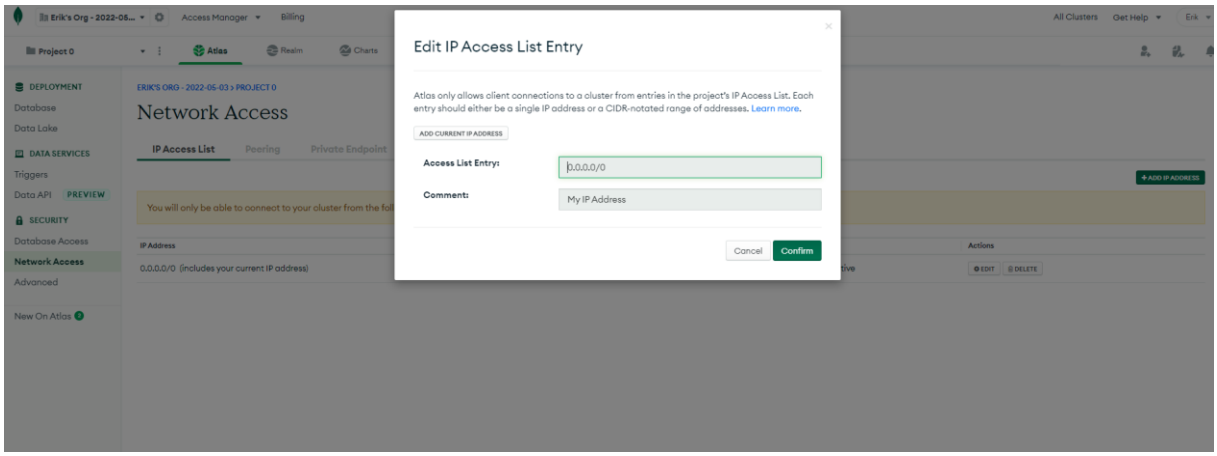


2. Povezava podatkovne baze MongoDB Atlas z našo aplikacijo

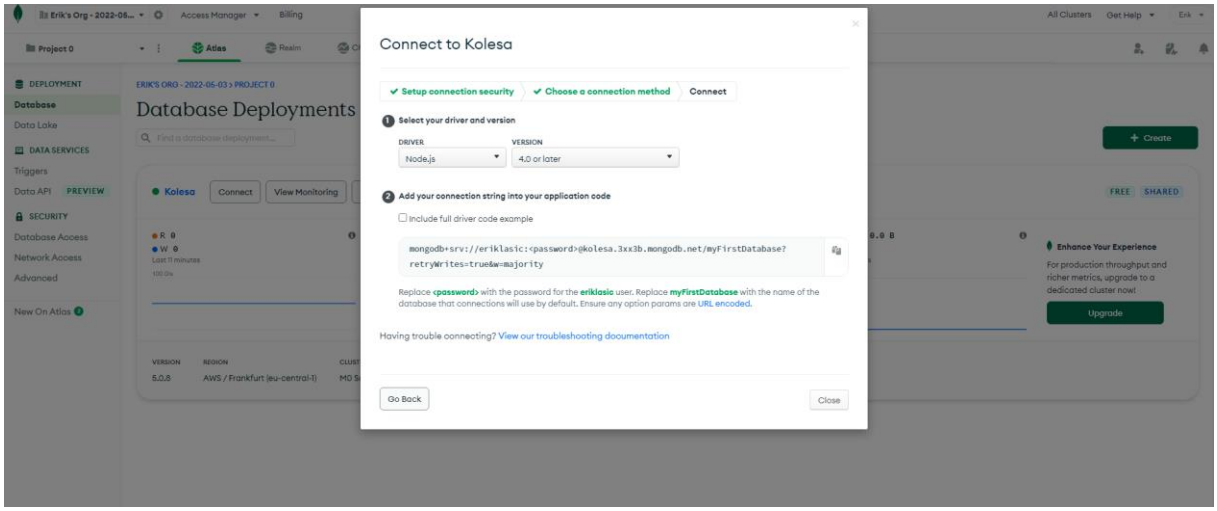
Sprva smo morali narediti »Cluster« na spletni strani MongoDB Atlas. MongoDB Atlas Cluster je ponudba NoSQL Database-as-a-Service v javnem oblaku (na voljo v Microsoft Azure, Google Cloud Platform, Amazon Web Services). To je upravljana storitev MongoDB in z le nekaj kliki lahko nastavimo delujočo gručo MongoDB, ki je dostopna iz našega spletnega brskalnika.



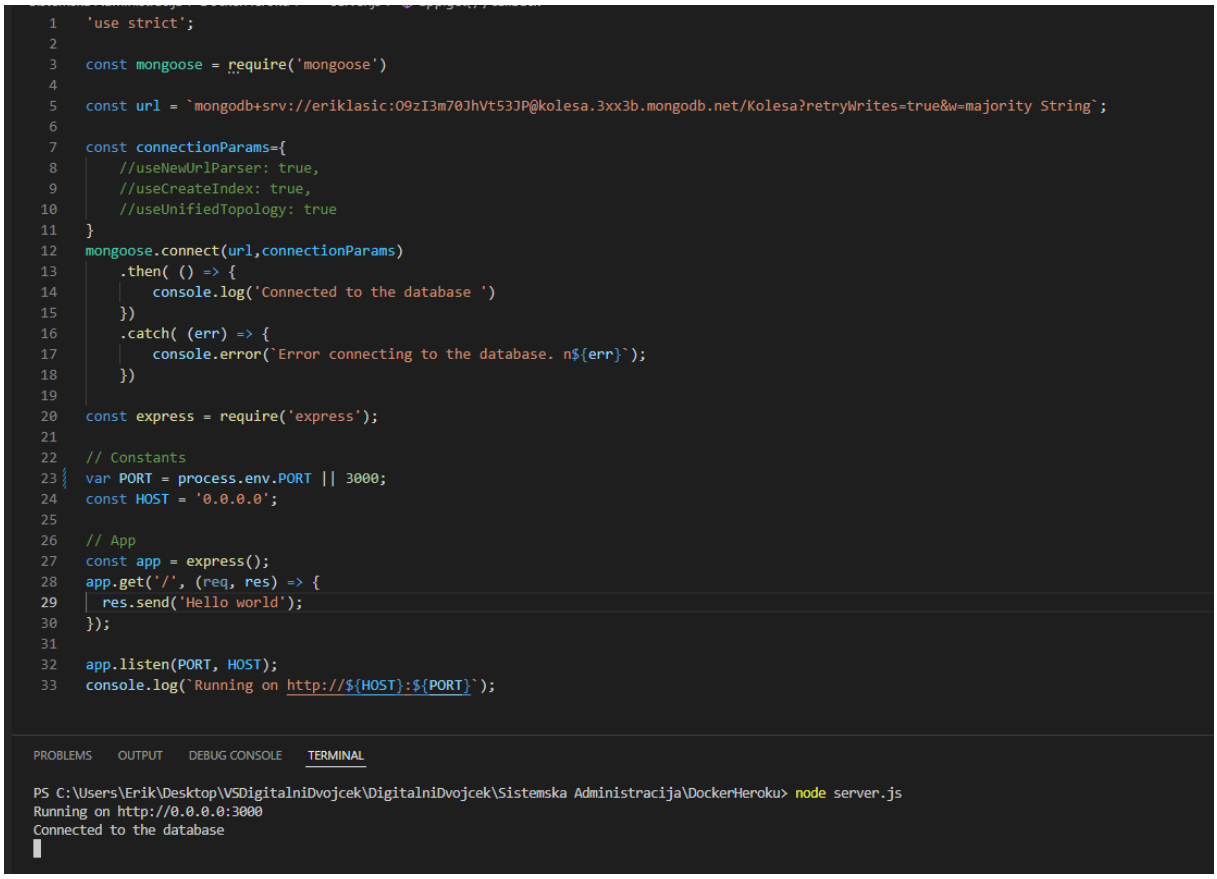
Nato smo morali pod zavihtkom Network Access omogočiti »Allow Access from Anywhere«, kar nastavi dostop na globalen. IP je zdaj 0.0.0.0/0



Zdaj smo se lahko lotili dejanske povezave naše aplikacije s podatkovno bazo. Pod Database gremo na Connect -> Connect your application, izberemo Node.js in skopiramo aplikacijski niz, ki ga postavimo v našo aplikacijo.



Na naši aplikaciji smo prvo morali v terminalu uporabiti ukaz npm install mongoose in v server.js napisali naslednje vrstice. Zamenjati je bilo potrebno <password> z geslom uporabnika in DatabaseName z imenom našega Clusterja, kar so Kolesa. Nato je bila naša aplikacija povezana s podatkovno bazo.



```
'use strict';

const mongoose = require('mongoose')

const url = `mongodb+srv://eriklasic:O9zI3m70JhVt53JP@kolesa.3xx3b.mongodb.net/Kolesa?retryWrites=true&w=majority String`;

const connectionParams={
    //useNewUrlParser: true,
    //useCreateIndex: true,
    //useUnifiedTopology: true
}
mongoose.connect(url,connectionParams)
    .then( () => {
        console.log('Connected to the database ')
    })
    .catch( (err) => {
        console.error(`Error connecting to the database. n${err}`);
    })

const express = require('express');

// Constants
var PORT = process.env.PORT || 3000;
const HOST = '0.0.0.0';

// App
const app = express();
app.get('/', (req, res) => {
  res.send('Hello world');
});

app.listen(PORT, HOST);
console.log(`Running on http://${HOST}:${PORT}`);
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\Erik\Desktop\VSDigitalniDvojcek\DigitalniDvojcek\Sistemska Administracija\DockerHeroku> node server.js
Running on http://0.0.0.0:3000
Connected to the database
```

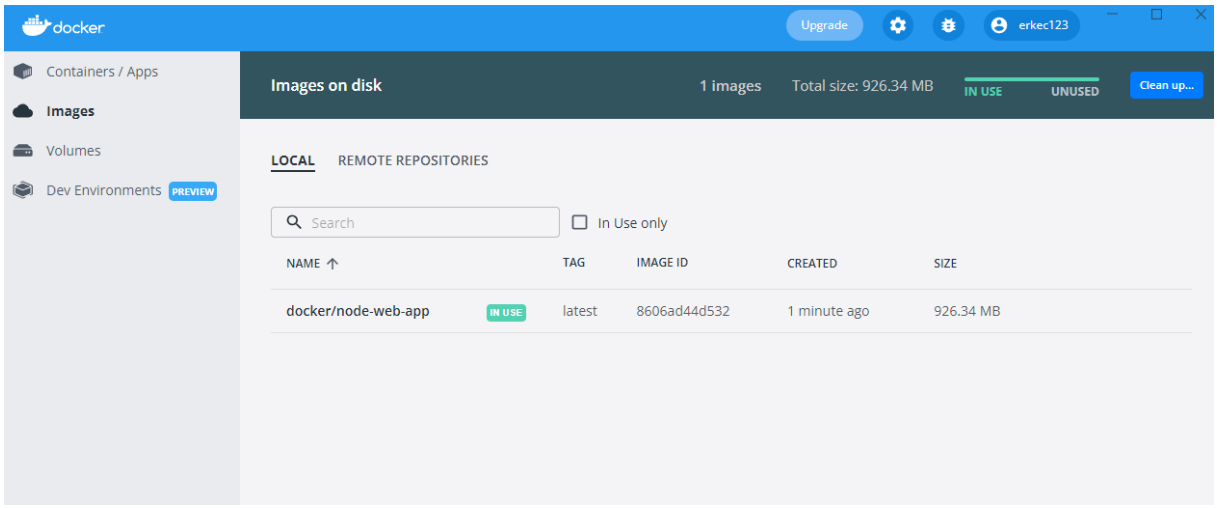
3. Namestitev aplikacije v okolju Docker s pomočjo dockerfile

Za izdelavo aplikacije moramo uporabiti datoteko Dockerfile. Dockerfile je preprosto besedilni skript navodil, ki se uporablja za ustvarjanje slike zaboja. Z ukazom »docker build . -t docker/node-web-app« smo zgenerirali sliko zaboja, ki se je nato pojavila v Docker-ju.

Dockerfile > ...

```
1   FROM node:16
2
3   # Create app directory
4   WORKDIR /usr/src/app
5
6   # Install app dependencies
7   # A wildcard is used to ensure both package.json AND package-lock.json are copied
8   # where available (npm@5+)
9   COPY package*.json ./
10
11  RUN npm install
12  # If you are building your code for production
13  # RUN npm ci --only=production
14
15  # Bundle app source
16  COPY . .
17
18  EXPOSE 3000
19  CMD [ "node", "server.js" ]
```

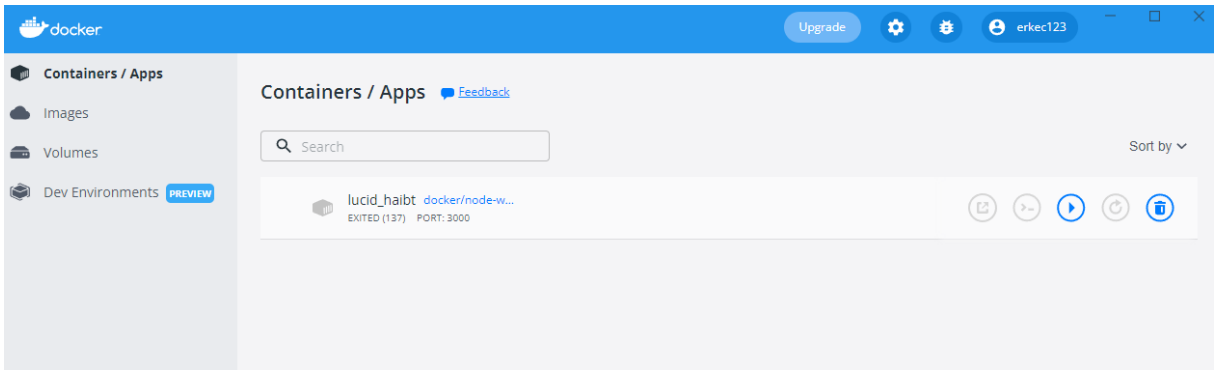
```
PS C:\Users\Erik\Desktop\VSDigitalniDvojcek\DigitalniDvojcek\Sistemska Administracija\DockerHeroku> docker build . -t docker/node-web-app
[+] Building 28.9s (11/11) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 439B
 => [internal] load .dockerignore
 => => transferring context: 67B
 => [internal] load metadata for docker.io/library/node:16
 => [auth] library/node:pull token for registry-1.docker.io
 => [internal] load build context
 => => transferring context: 1.70kB
 => [1/5] FROM docker.io/library/node:16@sha256:a6c217d7c8f001dc6fc081d55c2dd7fb3fefe871d5aa7be9c0c16bd62bea8e0c
 => CACHED [2/5] WORKDIR /usr/src/app
 => [3/5] COPY package*.json ./
 => [4/5] RUN npm install
 => [5/5] COPY . .
 => exporting to image
 => => exporting layers
 => => writing image sha256:76a44500ed245fdf65c76c0dffc2627050d553c9aeaee4f1da680cdec1f7d423
 => => naming to docker.io/docker/node-web-app
```



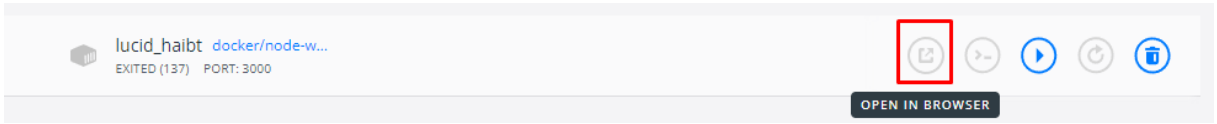
Zdaj je še potrebno zagnati zaboj naše aplikacije, to storimo z ukazom »docker run -p 3000:3000 -d docker/node-web-app, -d zastavica pomeni, da naš zaboj zaženemo v ločenem načinu in -p pomeni, da ustvarimo preslikavo med vrati gostitelja, ki so 3000 in vrati zaboja, ki so prav tako 3000

```
PS C:\Users\Erik\Desktop\VSDigitalniDvojcek\DigitalniDvojcek\Sistemska Administracija\DockerHeroku> docker run -p 3000:3000 -d docker/node-web-app
2a017888489e449b3fbf6fe9a8f4cb44e6069e9424b06824225c12a224650e83
PS C:\Users\Erik\Desktop\VSDigitalniDvojcek\DigitalniDvojcek\Sistemska Administracija\DockerHeroku>
```

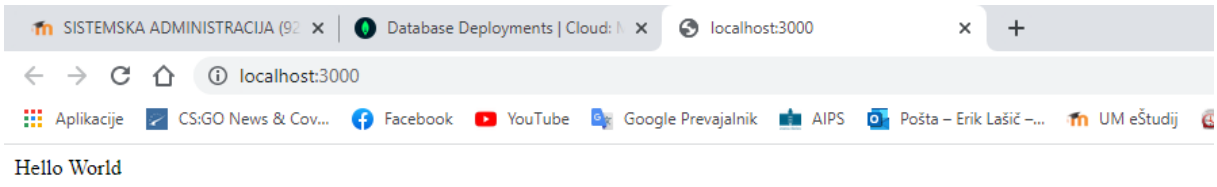
Ko se ukaz zažene se nam pojavi zaboj v Docker-ju.



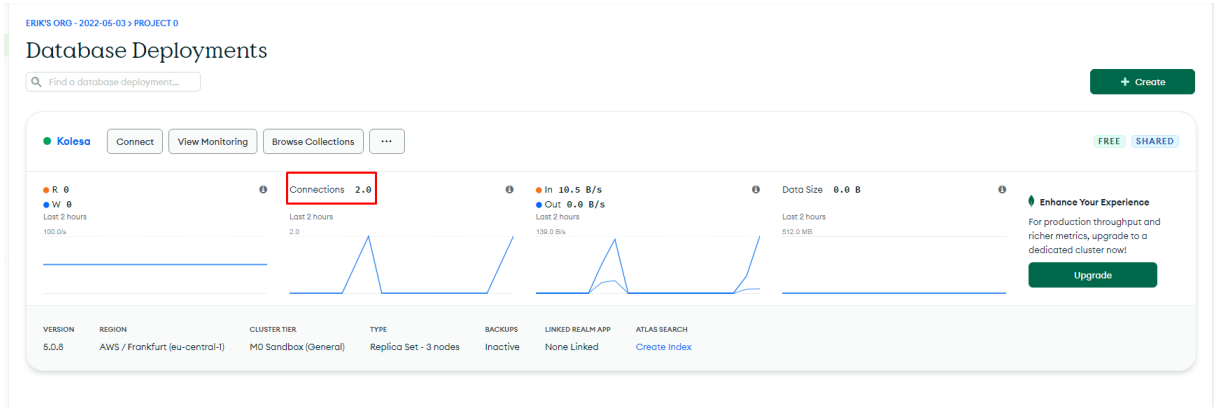
Če zaboj zaženemo, lahko preko brskalnika na naslovu http://localhost:3000 dostopamo do naše aplikacije ali pa stisnemo »Open in browser« in nam aplikacijo samo odpre



Tako zgleda naša aplikacija

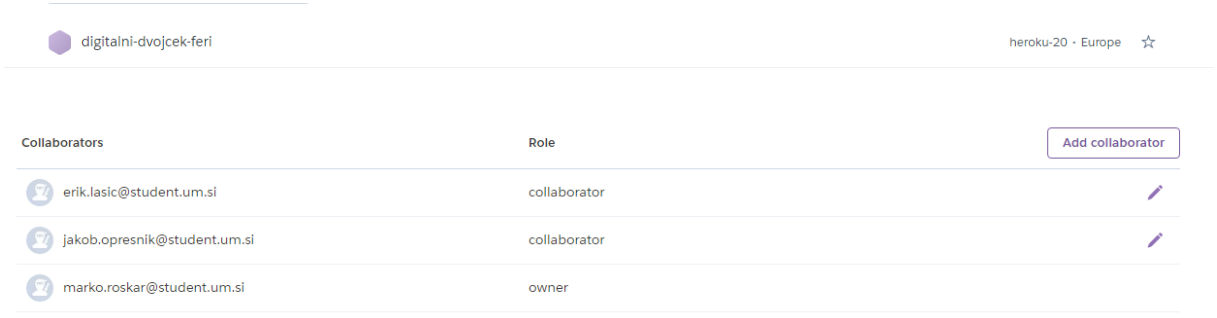


Na podatkovni bazi MongoDB Atlas lahko vidimo, da je povezava bila uspešno vzpostavljena.



4. Namestitev aplikacije na storitvi Heroku

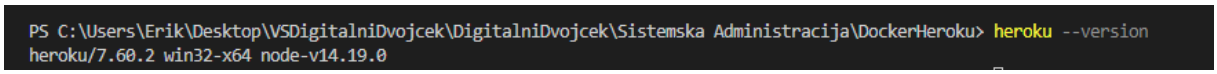
Naš vodja je ustvaril novo aplikacijo na Heroku



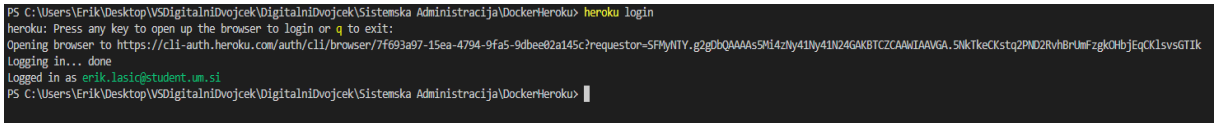
4.1 Heroku Container Registry

Heroku Container Registry omogoča, da svoje slike Dockerja namestimo v Heroku.

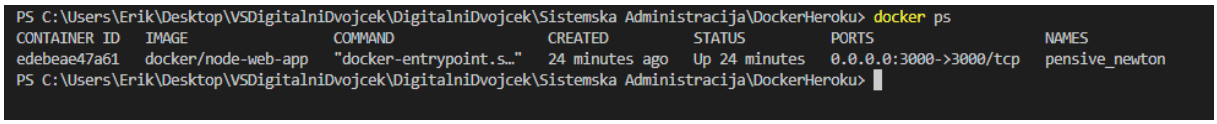
Prvo je potrebno namestiti Heroku CLI, za katerega potrebujemo nameščen Git.



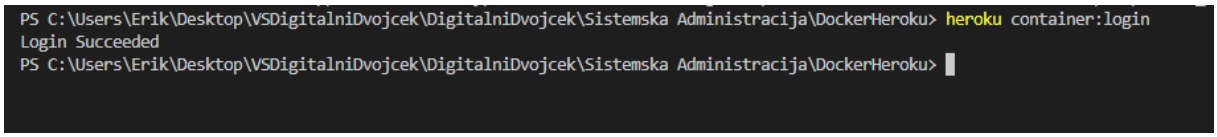
Da pričnemo z nameščanjem aplikacije s pomočjo »Container Registry« moramo prvo uporabiti ukaz »heroku login«, kar nas preusmeri na spletno stran Heroku, kjer potrdimo dostop.



Zdaj preverimo, če je Docker pravilno nameščen lokalno z ukazom »docker ps«



Nato se lahko prijavimo v »Container Registry« z ukazom »heroku container:login«



Zdaj lahko potisnemo svojo aplikacijo, ki temelji na Dockerju, tako da zgradimo Dockerfile v trenutnem direktoriju in potisnemo sliko. Za to uporabimo ukaz »heroku container:push web«

```
PS C:\Users\Erik\Desktop\VSDigitalniDvojcek\DigitalniDvojcek\Sistemska Administracija\DockerHeroku> heroku container:push web
=== Building web (C:\Users\Erik\Desktop\VSDigitalniDvojcek\DigitalniDvojcek\Sistemska Administracija\DockerHeroku\Dockerfile)
[+] Building 7.1s (11/11) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 439B
 => [internal] load .dockerignore
 => => transferring context: 67B
 => [internal] load metadata for docker.io/library/node:16
 => [auth] library/node:pull token for registry-1.docker.io
 => [internal] load build context
 => => transferring context: 60.01kB
 => [1/5] FROM docker.io/library/node:16@sha256:a6c217d7c8f001dc6fc081d55c2dd7fb3fefe871d5aa7be9c0c16bd62bea8e0c
 => CACHED [2/5] WORKDIR /usr/src/app
 => CACHED [3/5] COPY package*.json ./
 => CACHED [4/5] RUN npm install
 => CACHED [5/5] COPY . .
 => exporting to image
 => => exporting layers
 => => writing image sha256:b23e015a06bd3a372967c7b1050e1fc7163f930a1c452a7d5d574af1dd6264b0
 => => naming to registry.heroku.com/digitalni-dvojcek-feri/web

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
=== Pushing web (C:\Users\Erik\Desktop\VSDigitalniDvojcek\DigitalniDvojcek\Sistemska Administracija\DockerHeroku\Dockerfile)
Using default tag: latest
The push refers to repository [registry.heroku.com/digitalni-dvojcek-feri/web]
5ed14a0dcfdf: Pushed
8b236a04615d: Pushed
016310243ba7: Pushed
ce2b441f27d1: Pushed
0fc8a3e8b32a: Pushed
99307ceff565: Pushed
5cc685c4cd61: Pushed
6fd97e423126: Pushed
ca58f1c44290: Pushed
957a6eed8d1f: Pushed
85fe00380881: Pushed
5d253e59e523: Pushed
b9fd5db9c9a6: Pushed
latest: digest: sha256:fa2a4f05812aefd26513e93fcf430592a3c283096ff2abf6d8e13224b1927aea size: 3049
Your image has been successfully pushed. You can now release it with the 'container:release' command.
PS C:\Users\Erik\Desktop\VSDigitalniDvojcek\DigitalniDvojcek\Sistemska Administracija\DockerHeroku>
```

Končno lahko razvijemo nove potisnjene slike, da ustvarimo našo aplikacijo z ukazom »heroku container:release web«

```
PS C:\Users\Erik\Desktop\VSDigitalniDvojcek\DigitalniDvojcek\Sistemska Administracija\DockerHeroku> heroku container:release web
Releasing images web to digitalni-dvojcek-feri... done
PS C:\Users\Erik\Desktop\VSDigitalniDvojcek\DigitalniDvojcek\Sistemska Administracija\DockerHeroku>
```

Da odpremo našo aplikacijo v brskalniku uporabimo ukaz »heroku open«